AUTOMATICALLY DETECTING AND DESCRIBING HIGH LEVEL ACTIONS WITHIN METHODS

Giriprasad Sridhara, Lori Pollock and K. Vijay-Shanker

Natural Language Program Analysis Laboratory
Computer and Information Sciences
University of Delaware
U.S.A

33rd IEEE/ACM International Conference on Software Engineering (ICSE)
Hawaii, USA: May 25 -27, 2011
Program Comprehension is not Easy

- Issue: Developers need to read a lot of code

```java
for (int x = 0, n = vAttacks.size(); x < n; x++) {
    WeaponAttackAction waa = vAttacks.elementAt(x);
    float fDanger = getExpectedDamage(g, waa);
    if (fDanger > fHighest) {
        fHighest = fDanger;
        waaHighest = waa;
    }
}
```

Goal: Automatically detect and describe high level actions

Generated Description After High Level Action Detection

```bash
// Find weapon attack action object // (in vectorAttacks) with highest expected damage
for (int x = 0, n = vAttacks.size(); x < n; x++) {
    // ...
}
```

Benefits:
- Programmers can focus on fewer & more abstract ideas
- Overcome issues with developer written descriptions:
  Not always present or maintained up to date

Giriprasad Sridhara
Our Contributions

**Developed Heuristics to Automatically:**

- *Detect* a group of statements that together implement a higher level algorithmic step (action)
- *Generate* a succinct English phrase describing the high level action

**Evaluated Precision, Prevalence, and Potential Use:**

Experienced developers *judged* high level action identification & description

*Automatically counted* prevalence of identification heuristics across 1000 open source projects

*Estimated* potential reduction in reading

Outline for the talk
Types of High Level Actions within a Method

- **A sequence** of statements that together represent a high level action:

```java
noFirewall = new JRadioButton("No firewall or proxy");
socksFirewall = new JRadioButton("SOCKS 4/5 Firewall");
webProxy = new JRadioButton("HTTP Web Proxy");

allButtons = new ButtonGroup();
allButtons.add(socksFirewall);
allButtons.add(webProxy);
allButtons.add(noFirewall);

socksFirewall.addActionListener(rad);
webProxy.addActionListener(rad);
noFirewall.addActionListener(rad);
```

- **Create RadioButtons**
- **Add RadioButtons to allButtons**
- **Add radioActionListener to RadioButtons**
Types of High Level Actions

- Conditional branches that perform similar actions with subtle variations (based on the condition)

```java
if ( type == PT_HTTP ){
    tester = new NetworkAdminHTTPTester(core, l);
} else if ( type == PT_TCP ){
    tester = new NetworkAdminTCPTester(core, l);
} else{
    tester = new NetworkAdminUDPTester(core, l);
}
```

create network admin protocol tester based on type

Based on Data Type of tester, we generate NetworkAdminProtocolTester instead of tester

Process: Lexicalization

Aim: Generate more descriptive phrases
Types of High Level Actions

- **Loop constructs** that implement common algorithms (e.g., find, count)

```java
for (DrawingView v : views) {
    if (v.getComponent() == c)
        return v;
}
```

Find (return) drawing view whose component equals given container

Techniques to generate relative clause

Signifies `container` is a parameter
Overview of Automated Process

INPUT:
Method M signature and body

Build required structural and linguistic program representations

Identify high level action(s) (Sequences, Conditionals, Loops)

Generate Phrase for each identified high level action

OUTPUT:
Method M with high level actions identified and described
Preprocessing:
Build Structural & Linguistic Program Representations

- Build Abstract Syntax Tree
- Construct Control Flow Graph
- Create Variable Def-Use Chains

Method M
signature & body

Split Identifiers into Words
[Hill et al 08]

Expand Abbreviated Words
[Hil et al 08]
Background: Describing Method Calls as Verb Phrases

Methods perform actions → Generate verb phrase (ASE 2010)

Verb Phrase → Verb → NounPhrase

Optional Prepositional Phrases

os.print(msg)

UD-GenPhrase (ASE 2010)

Semantics:

Action
Theme
Secondary Argument

/* print message to output stream*/
 Detecting High Level Actions in Sequences

- Our target: A sequence of possibly different callees with the same action

```java
public void reset() {
    removeAll();
    buildGameMenu();
    buildViewMenu();
    buildToolsMenu();
    buildColopediaMenu();
    // --> Debug
    if (FreeCol.isInDebugMode()) {
        add(new DebugMenu(freeColClient));
    }
    update();
}
```
Challenge:

- A sequence of same callees (i.e., same action) need not form a high level action

Need to consider the theme of the action as well

```java
contentPane.add(endedPanel);
contentPane.add(bidPanel);
contentPane.add(endingPanel);
contentPane.add(OKButton);
```

Add panels to content pane

Add panels & OKButton to content pane

Add components to content pane

Too Generic & Vague compared to “panels”

Component: Formal Parameter Type Name for add

Detecting these situations helps generate a more precise & concise description
Our Approach to Detecting HL Action in Sequences:

**INPUT: Statements S₁, S₂**

- VP₁ = generateVerbPhraseFor(S₁);
- VP₂ = generateVerbPhraseFor(S₂);
- IF sameVerb(VP₁, VP₂) AND
- IF sameHeadWord(NP₁, NP₂) AND
- IF present(PP₁, PP₂) AND
- IF samePreposition(PP₁, PP₂) THEN

S₁ & S₂ can be integrated to form a High Level Action in Sequence

**Example:**

- S₁: onelineBox.add(first);
- VP₁: add first component to onelineBox
- S₂: onelineBox.add(second);
- VP₂: add second component to onelineBox

- onelineBox.add(first);
- onelineBox.add(second);

**Integrable**

**In a method:**
- We find if pairs of statements are integrable, starting with the first statement
- We build a fragment of such integrable statements of maximum possible length
Synthesizing Description: Example

After we find a sequence, we need to synthesize a description

**VP1:** Add first component to onelineBox

**VP2:** Add second component to onelineBox

**Variations:**
- **Noun Phrases** correspond to fields of a class
- **Noun Phrases of the Prepositional Phrase** share the same head word
Finding High Level Actions in Conditionals: Issues

- Integrating statements along the different branches
  - Similar heuristics as in finding sequence high level actions
  - Additionally, in every branch, we can have
    - A return statement
    - Assignment to the same variable
    - Antonym actions (opposites)
      The above will typically not occur in a sequence
  - Strategies for the above

```java
switch (movementType) {
    if (type == PT_HTTP) {
        tester = new NetworkAdminHTTPTester(core, 1);
    } else if (type == PT_FILE) {
        if (os.startsWith("mac")) {
            app = new DefaultOSXApplication();
        } else {
            else if (os.startsWith("win")) {
                // app = new DefaultMDIApplication();
                app = new DefaultSDIApplication();
            } else {
                app = new DefaultSDIApplication();
            }
        }
    }
}
```
Finding High Level Actions in Conditionals: Example

```java
if (os.startsWith("mac")) {
    app = new DefaultOSXApplication();
} else if (os.startsWith("win")) {
    // app = new DefaultMDIApplication();
    app = new DefaultSDIApplication();
} else {
    app = new DefaultSDIApplication();
}
```

Create application, based on what os starts with

Usage of relative pronoun "what": Because predicate "starts with" begins with a verb in the 3rd person singular form

More information in paper
Finding High Level Actions in Loops

- Many common algorithmic actions involve loops: Ex: sum, find, count
- Examined many loops across many open source projects
- As proof of concept, developed templates for identifying, and heuristics for synthesizing phrases:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Find maximum</strong></td>
<td>Find an item in a collection with a maximum (minimum) value computed using some specified criteria</td>
</tr>
<tr>
<td><strong>(minimum)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Count</strong></td>
<td>Count items in a collection that meet a specified criteria</td>
</tr>
<tr>
<td><strong>Contains</strong></td>
<td>Check if a collection contains an item that has some specified property</td>
</tr>
<tr>
<td><strong>Find</strong></td>
<td>Find an item in a collection that meets some specified criteria</td>
</tr>
<tr>
<td><strong>Copy</strong></td>
<td>Copy one or more items in a collection that have a desired attribute(s) to another collection</td>
</tr>
</tbody>
</table>
Example: Template for Find Max/Min

**ALGORITHM TEMPLATE:**

1. Initialize variable, max, to 0 or -1
2. Initialize variable, maxItem, to null
3. Loop over each item i in a Collection c
   
   4. Set variable, current, with the return value from a method-call which uses i as a parameter
   
   5. IF (current > max) //also: max < current
   
   6. update max to current
   
   7. update maxItem to i

**SYNTHESIS TEMPLATE:**
find <item> (in <collection>) with <adjective> <criteria>
Find Maximum: Example

```java
for (int x = 0, n = vAttacks.size(); x < n; x++) {
    WeaponAttackAction waa = vAttacks.elementAt(x);
    float fDanger = getExpectedDamage(g, waa);
    if (fDanger > fHighest) {
        fHighest = fDanger;
        waaHighest = waa;
    }
}
```

SYNTHESIS TEMPLATE: find <item> (in <collection>) with <adjective> <criteria>

Synthesized: find weapon attack action object (in vector attacks) with highest expected damage

Matches the Find maximum template
Evaluating Effectiveness of our Approach

- **Prevalence**
  - How prevalent are the high-level actions identified by our heuristics in Java software?

- **Precision**
  - How precisely do we identify code fragments that implement a high level action?
  - How well does the synthesized description represent the high level action?
“Prevalence” Study

- Question: How prevalent are the high-level actions identified by our heuristics in Java software?

- Setup
  - 1000 open source Java projects from Sourceforge
    - Cumulatively about 1.2 million methods
    - Median: 314 methods
    - Maximum: 144,604 methods
  - Ran prototype: gathered frequency of detection
    - Sequence
    - Conditional
    - Loop high level actions
Prevalence Results

- **Sequence**
  - Typically not present in very small methods
  - Examined methods with at least 10 statements
    - 155, 289 such methods
  - 17, 205 (11%) methods have a sequence high-level action

- **Conditional**
  - 144,562 if-else statements
  - 58,439 (40%) had an identified high level action
  - 17, 940 switch statements
  - 4,319 (24%) had an identified high level action

- **Loop**
  - 82,402 loops iterating over all items in a collection
  - 12,524 (15%) implement one of the algorithmic patterns
“Precision” Study

Questions:
- How precisely do we identify code fragments that implement a high level action?
- How well does the synthesized description represent the high level action?

Setup:
- **15 Java Programmers as Evaluators:**
  - Post Doctoral Researchers & PhD / MS Students
  - Mostly advanced programming skills (self-identified)
  - 7 of them have software industry experience
- **15 large open source Java projects**
- Evaluated synthesized descriptions of **75 fragments**
  25 each of sequences, conditionals and loops
- **Subjectivity:** 3 human opinions per fragment
- **Avoided biasing evaluators**
  Did not provide a definition or an example of a high level action
Precision Results: Programmer Opinions on...

- Proposition 1: The fragment of statements from lines : X to Y reflects a high level action that could be expressed as a succinct phrase by a human. i.e.,
  - There are no other statement(s) in the method that you would include in the fragment, AND
  - There are no statement(s) in the fragment that you would exclude from the fragment.

Note: Entire method was shown to the evaluators

<table>
<thead>
<tr>
<th>Response Type (Likert Scale)</th>
<th># Responses</th>
<th>% of Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly Agree</td>
<td>192</td>
<td>85</td>
</tr>
<tr>
<td>Agree</td>
<td>21</td>
<td>9</td>
</tr>
<tr>
<td>Neither agree nor disagree</td>
<td>5</td>
<td>2.5</td>
</tr>
<tr>
<td>Disagree</td>
<td>5</td>
<td>2.5</td>
</tr>
<tr>
<td>Strongly Disagree</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>225</td>
<td>100</td>
</tr>
</tbody>
</table>

75 fragments and 225 Individual Responses
### Precision Results: Programmer Opinions on...

- **Proposition 2:** The description represents an abstraction of the block

<table>
<thead>
<tr>
<th>Response Type</th>
<th># Responses</th>
<th>% of Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly Agree</td>
<td>165</td>
<td>73</td>
</tr>
<tr>
<td>Agree</td>
<td>37</td>
<td>16</td>
</tr>
<tr>
<td>Neither agree nor disagree</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Disagree</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Strongly Disagree</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>225</td>
<td>100</td>
</tr>
</tbody>
</table>

75 fragments and 225 Individual Responses

*Good results in identifying and describing high level actions*
Improving Client Tools

- Automatic Descriptive Comment Generation
  - Comments within a method
  - Leading method summary comments
- Extract Method Refactoring
- Renaming suggestions
  - Methods
  - Local Variables
- Traceability Recovery
- Concern Location
Putting Other Work into Perspective

- **Beacons**: Signal something to the reader
  [Brooks 83, Storey 05]
  - Not automated

- **Micro Patterns** [Gil and Maman 05]
  - 27 patterns at the level of a class and not methods

- Extract Topic Words and Phrases
  [Maskeri et al 08]

- Cluster program elements sharing similar words
  [Kuhn et al 07]
  - Not necessarily high level actions
Summary Comment Generation for Java Methods: [ASE 2010]*
- Identified *individual* statements for a method summary
- Generated phrases for each statement in isolation
- *This work enables more succinct summaries and improved summary content selection*

Parameter comment generation and Integrating with method summaries: [ICPC 2011: To Appear]

* ACM SIGSOFT Distinguished Paper
Future Work

- Utilize high level action identification and description for more succinct method summary comments

- Develop heuristics to identify more high level actions

- Evaluate in a client tool e.g., ExtractMethod Refactoring
Summary of Contributions

- First technique to
  - Identify sequences, conditionals and loops that can be abstracted as a high-level action, and
  - Synthesize an English description of the abstraction
- Experienced developers provided positive feedback
- Good prevalence across 1000 open source projects
- Many Potential Uses
  - Automatic Comment Generation
  - Extract Method Refactoring
  - Suggest better method and variable names
  - Traceability recovery & Concern location

Acknowledgement: NSF Grants CCF-0702401 & CCF-0915803
Helper Slides: Index

- Other loop patterns starting with count including synthesis rules and example: 32 to 40
- Client tools: ExtractMethod, Renaming, Traceability: 41 to 47
- Variable lexicalization: 48
- Sequences: Other cases examples: 49 to 51
- Conditionals: Return rules & example: 52 53
- Conditionals: Guarding expressions: 54 to 58
- Comment studies: 59
- getHighestExpectedDamage: example 1: 60
v = 0 (initialization)
for each <item> in a <collection>
  if (<item> used in a boolean valued expression)
    v++ (or v = v + 1) [v is not used as an array index]
end for

use v outside (after) the above loop (final use)
otherwise v is not a count but an index used to iterate within the loop.

Note: The "if" above is optional.
When the "if" is not present:

if the 'count' variable, v, does not have the word "count" in it,
    Phrase =/* count v in <collection> */
Else if the 'count' variable, v, has multiple words with one of the words being 'count',
    Phrase =/* count <var name words excluding the count word> */
Else if the count variable has only the word 'count' or 'total' in it,
    Phrase = =/* count v in <collection> */
When the "if" is present:
- We have one phrase: <subject> <predicate> <object> for the IF expression and
- Another phrase for the count: count plural(<item>) in <collection>.

When the <subject> or <object> is the same as <item>, we can combine these phrases into one smooth phrase using a relative clause (by using a relative pronoun for <item>).

Such a phrase sounds more natural i.e., looks as if it were written by a human.

The specific rules follow
Count Pattern : Synthesis Template

In the phrase for the "if" expression (of the form <subject> <predicate> <object>)

Rule 1:

IF <subject> = <item> AND <predicate> begins with a Verb 3rd Person Singular (e.g. equals, startsWith) THEN
   Phrase with relative clause (relative pronoun "which" for <item>):
   /* count plural(lastWord(<item>)) in <collection>,
       which <singular of the V3PS> <remainder of the predicate> <object> */

Rule 2:

IF <subject> = <item> AND <predicate> begins with an auxiliary verb (e.g. is, has) THEN
   Phrase with relative clause (relative pronoun "which" for <item>):
   /* count plural(lastWord(<item>)) in <collection>, which <plural of the aux verb>
      <remainder of predicate> */

Rule 3:

IF <subject> is of the form " X of <item>" (i.e., <subject> is a possessive of <item>)
   AND
   (<predicate> begins with a V3PS (e.g. equals) OR <predicate> begins with an aux verb) THEN
   Phrase with relative clause (relative pronoun "whose" for <item>)
   /* count <Plural of the last word of <item> > in <collection>, whose X <predicate> <object> */
Rule 2 applies:
IF <subject> = <item> AND <predicate> begins with an auxiliary verb (e.g. is, has) THEN
Phrase with relative clause (relative pronoun "which" for <item>):
/* count plural(lastWord(<item>)) in <collection>,
which <plural of the V3IRR> <remainder of predicate> */

Count vuze activities entries in allEntries, which are not read

Simplified from:
Count vuze activities entries in allEntries, such that vuze activities entries are not read
Find Pattern : Template

Initialize <found-item> to null

for each <item> in a <collection>
   if (<item> used in a boolean valued expression)
      <found-item> = <item>;
      break;// could also be "return <found-item>"
   end if
end for

use <found-item> outside the above loop (unless there was a return instead of the break)

Phrase with relative clause formed by combining the "<subject> <predicate> <object>"
phrase of the IF expression and the "for each <item> in <collection>" phrase for the loop

Relative pronoun is determined as in the "count" pattern case.

The final phrase is also formed as in the "count" pattern case, except that there are no plurals or singular of the predicate.
Find Pattern : Example

```java
for (DrawingView v : views) {
    if (v.getComponent() == c)
        return v;
}
```

Matches the Find Template
i.e.,
Find <item> (in <collection>),
such that <criteria>

Item = DrawingView
Collection = Views

Criteria:

- **subject**
- **predicate**
- **object**

Find drawing view in views
such that component of drawing view equals the given container

**Rule 3 applies:**

IF <subject> is of the form " X of <item>" (i.e., <subject> is a possessive of <item>)
AND <predicate> begins with a Verb 3rd Person Singular (e.g. equals) THEN
Phrase with relative clause (relative pronoun "whose" for <item>)
/* find <item> > (in <collection>), whose X <predicate> <object> */

Find drawing view,
whose component equals the given container

Giriprasad Sridhara
CONTAINS (i.e. find if)

boolean found = false;

for each <item> in a <collection>
    if (<item> used in a boolean valued expression)
        found = true;
        break; // could also be "return found"
    end if
end for

use found outside the above loop

Phrase (formed by combining the "<subject> <predicate> <object>" phrase of the IF expression and the "for each <item> in <collection>" phrase for the loop)

/* find if any <item> in <collection> <predicate> <object> */
COPY ITEMS FROM ONE COLLECTION TO ANOTHER COLLECTION

for each <item> in a <collection>
if (<item> used in a boolean valued expression)
   add <item> to <collection-2>  // adding is copying
end if
end for

Phrase with relative clause formed by combining the "<subject> <predicate> <object>"
phrase of the IF expression and the "add <item> to <collection-2>" phrase for the loop.

Phrase formation rules similar to the "count" pattern.
Client Tools: Extract Method Refactoring

- **Steps in ExtractMethod Refactoring:**
  - Identify refactoring candidate to extract into a separate method
  - Provide a descriptive name for the extracted method
  - Identify input parameters and return type for extracted method
  - Create a new method
  - Replace fragment in the original method by a call to the extracted method

*Our System can Help*

*Support Available for Developer (Ex: Eclipse)*
Use in Extract Method Refactoring

Alternate to ExtractMethod refactoring: Internal comments

```java
public static void main(String[] args) {
    Application app;
    String os = System.getProperty("os.name").toLowerCase();
    if (os.startsWith("mac")) {
        app = new DefaultOSXApplication();
    } else if (os.startsWith("windows")) {
        // app = ...
    }

    SVGApplicationModel model = new SVGApplicationModel();
    model.setDifferentAttributes(model);
    app.setModel(model);
    app.launch(args);
}
```

Set different attributes of SVG Application Model

Giriprasad Sridhara
Renaming Suggestion

- `findDrawingView` instead of `findView` [better theme]

```java
public DrawingViewfindView(Container c) {
    for (DrawingView v : views) {
        if (v.getComponent() == c) {
            return v;
        }
    }
    return null;
}
```
Renaming Suggestion

- The local variable “count” is not really “counting”
- It is used as an index into the array
- A typical “count” variable will be used outside the array

```java
public Target[] getTargets()
{
    Target[] targs = new Target[targets.size()];
    int count = 0;
    for (Enumeration e = targets.getObjects(); e.hasMoreElements();)
    {
        targs[count++] = Target.getInstance(e.nextElement());
    }
    return targs;
}
```
Renaming Suggestion

- findSearch instead of getSearch [better action]
  - Find searcher whose name equals given name

```java
public Searcher getSearchBy_name(String name) {
    for (Iterator it = _searches.iterator(); it.hasNext(); ) {
        Searcher search = (Searcher)it.next();
        if (search.getName().equals(name)) return search;
    }

    return null;
}
```
Provide new words (verb) that could be used in search/traceability recovery

Can also help in ranking results higher

// Find weapon attack action object
// (in vectorAttacks) with highest expected damage
for (int x = 0, n = vAttacks.size(); x < n; x++) {
Renaming Suggestion

- findHook instead of getHook [better action]
  - Find other permanent node hook (in activated hooks) whose name equals given hook name

```java
public PermanentNodeHook getHookInNode(MindMapNode node, String hookName) {
    // search for already instanciated hooks of this type:
    for (Iterator i = node.getActiveActivatedHooks().iterator(); i.hasNext();)
    {
        PermanentNodeHook otherHook = i.next();
        if (otherHook.getName().equals(hookName)) {
            // there is already one instance.
            return otherHook;
        }
    }
    return null;
}
```
Background: Variable Lexicalization (ASE 2010)

print(current);

/* print current */

/* print current document */

But what is current?

- While reading a method, the context will tell what ‘current’ is, but outside of the body we need to fill in the context.
- Key Insight: Type information typically provides missing context.

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Variable Name</th>
<th>UDGenSubPhrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallFrame</td>
<td>parentFrame</td>
<td>parent call frame</td>
</tr>
</tbody>
</table>

Selectable is an adjective

Frame is not repeated

- Generation:
  - part of speech of words in type and variable names
  - overlap between type & variable name
Sequences: Other examples

```java
public Searcher buildSearch(long id, String type,
    Searcher s = newSearch(type);
s.setId(id);
s.setName(name);
s.setSearch(search);
s.setServer(server);
s.setPeriod(period);
s.setCurrency(currency);
    return s;
}
```

Set different attributes of Searcher
if (readModel != null) {
    controller.setModel(piece.getModel());
    controller.setModelRotation(piece.getModelRotation());
    controller.setBackFaceShown(piece.isBackFaceShown());
    controller.setName(piece.getName());
    controller.setCategory(piece.getCategory());
    controller.setWidth(piece.getWidth());
    controller.setDepth(piece.getDepth());
    controller.setHeight(piece.getHeight());
    controller.setMovable(piece.isMovable());
    controller.setDoorOrWindow(piece.isDoorOrWindow());
    controller.setElevation(piece.getElevation());
    controller.setColor(piece.getColor());
    controller.setIconYaw(piece.getIconYaw());
    controller.setProportional(piece.isProportional());
}

Set all attributes of imported furniture wizard controller
Sequences : Other examples

- **Current**: Add buttons to button box
  - **Update**: make buttons. Add buttons to button box

```javascript
buttonBox.add(jsc.makeButton("Search", "Execute"));
buttonBox.add(jsc.makeButton("New"));
buttonBox.add(jsc.makeButton("Edit", "Edit Search"));
buttonBox.add(jsc.makeButton("Enable"));
buttonBox.add(jsc.makeButton("Disable"));
```

- **Current**: Not identified as a sequence HL Action as the calls are not contiguous / in same block
  - **Update**: create chart panels. add chart panels to stat view

```javascript
ChartPanel cp1 = createStyleRepartition();
if ( cp1!= null) add(cp1,"0,0");
ChartPanel cp2 = createCollectionSize();
if ( cp2!= null) add(cp2,"0,2");
ChartPanel cp3 = createTrackNumber();
if ( cp3!= null) add(cp3,"2,2");
ChartPanel cp4 = createDeviceRepartition();
if ( cp4!= null) add(cp4,"2,0");
```
Conditionals: Return : Rules

Let the current method be M (i.e., method having the return statement(s))

IF return is present in the THEN and ELSE branches (or in each case of a SWITCH excluding the default case) AND

IF M is not a void-return method [to avoid integrating “return ;” statements] AND

IF the return expression is not NULL [to avoid integrating “return null;” statements] AND

IF action(M) = “get” OR IF semantic-getter(M) (i.e., theme (M) = return-type(M) OR same head words)

THEN

Integrate the return statements.

Integrated Phrase = return theme(M)
Conditionals: Return : Example

```java
switch (movementType) {
    case IEntityMovementType.MOVENONE: return "N";
    case IEntityMovementType.MOVEWALK: return "W";
    case IEntityMovementType.MOVERUN: return "R";
}
```

Theme of the enclosing method: movement abbreviation
Action of the enclosing method: get

Return movement abbreviation
Conditionals: Integrating the Guarding Expressions

Heuristic:
IF subject and predicate of each clause is same THEN
Abstraction = based on what subject predicate

Based on what os starts with

Usage of relative pronoun “what”:
Because predicate “starts with” begins with a verb in the 3rd person singular form
Conditionals: Integrating the Guarding Expressions

Heuristic:

IF subject and predicate of each clause is same AND predicate is “equals” THEN Abstraction = based on subject

Based on what lookAndFeel string equals

Based on lookAndFeel string

Set LookAndFeel of UIManager, based on lookAndFeel string
Conditionals: Integrating the Guarding Expressions

```java
if (source instanceof Europe) {
    showEuropePanel();
} else if (source instanceof Colony) {
    showColonyPanel((Colony) source);
} else if (source instanceof WorkLocation) {
    showColonyPanel(((WorkLocation) source).getColony());
}
```

Heuristics:
IF subject and predicate of each clause is same AND predicate is "instanceOf" THEN Abstraction = based on type of subject

based on type of source FreeColGameObject

show panel, based on type of source FreeColGameObject
Conditionals: Integrating the Guarding Expressions

```java
if (noFirewall.isSelected()) {
    firewallState = "none";
} else if (socksFirewall.isSelected()) {
    firewallState = "firewall";
} else if (webProxy.isSelected()) {
    firewallState = "proxy";
}
```

Generate a phrase for each conditional expression [ASE 2010]

Heuristic:
IF headWord(subject) and predicate of each clause is same THEN Abstraction=based on which head-word(subject) predicate

Based on which RadioButton is selected

Usage of relative pronoun “which”:
Because predicate “is” begins with an auxiliary verb is

Update firewallState String, based on which RadioButton is selected
Conditionals: Integrating the Guarding Expressions

```java
if (player.isGhost()) {
    playerDisplay.append(Messages.getString("PlayerListDialog.player_ghost"));
} else if (player.isObserver()) {
    playerDisplay.append(Messages.getString("PlayerListDialog.player_observer"));
} else if (player.isDone()) {
    playerDisplay.append(Messages.getString("PlayerListDialog.player_done"));
}
```

Heuristic:
IF subject is the same AND predicate is not the same but begins with an auxiliary verb (is/has/was) THEN Abstraction=based on property of subject

Based on property of player

Append string to player display string, based on property of player
Comment Studies

- Many studies have shown the utility of comments (including comments within a method) [Tenny 88, Takang et al 96]
- Programmers tend to comment more in an already well commented system [Marin 05]
/** Returns the weapon attack out of a list that has the highest expected damage */

public static WeaponAttackAction getHighestExpectedDamage(IGame g, Vector vAttacks) {
    float fHighest = -1.0f;
    WeaponAttackAction waaHighest = null;
    for (int x = 0, n = vAttacks.size(); x < n; x++) {
        WeaponAttackAction waa = (WeaponAttackAction) vAttacks.elementAt(x);
        float fDanger = getExpectedDamage(g, waa);
        if (fDanger > fHighest) {
            fHighest = fDanger;
            waaHighest = waa;
        }
    }
    return waaHighest;
}
Types of High Level Actions within a Method

- A *sequence* of statements that together represent a high level action: PUT the `add(Button), addActionListener`.

```java
public void reset() {
    removeAll();
    buildGameMenu();
    buildViewMenu();
    buildToolsMenu();
    buildColopediaMenu();

    // --> Debug
    if (FreeCol.isInDebugMode()) {
        add(new DebugMenu(freeColClient));
    }
    update();
}
```

Build Menus
Types of High Level Actions within a Method

- A sequence of statements that together represent a high level action:

```java
noFirewall = new JRadioButton("No firewall or proxy");
socksFirewall = new JRadioButton("SOCKS 4/5 Firewall");
webProxy = new JRadioButton("HTTP Web Proxy");

allButtons = new ButtonGroup();
allButtons.add(socksFirewall);
allButtons.add(webProxy);
allButtons.add(noFirewall);

socksFirewall.addActionListener(rad);
webProxy.addActionListener(rad);
noFirewall.addActionListener(rad);
```
Our Approach to Detecting HL Action in Sequences:

**INPUT: Statements S₁, S₂**

VP₁ = generateVerbPhraseFor(S₁);
VP₂ = generateVerbPhraseFor(S₂);

IF sameVerb(VP₁, VP₂) AND

IF sameHeadWord(NP₁, NP₂) AND

IF present(PP₁, PP₂) AND

IF samePreposition(PP₁, PP₂) THEN

S₁ & S₂ can be integrated to form a High Level Action in Sequence

**Example:**

S₁: contentPane.add(bidPanel);

VP₁: add bid panel to content pane

S₂: contentPane.add(endingPanel);

VP₂: add ending panel to content pane

contentPane.add(bidPanel);
contentPane.add(endingPanel);

Integrable

**In a method:**

• *We find if pairs of statements are integrable, starting with the first statement*

• *We build a fragment of such integrable statements of maximum possible length*
After we find a HL Action in a Sequence, we need to synthesize a description.

VP1: Add bid panel to content pane

VP2: Add ending panel to content pane

Add panels to content pane

Common Verb

Plural of Common Head Word of Noun Phrases

Common Preposition

Common NounPhrase of Prepositional Phrases

Variations:

- *Noun Phrases correspond to fields of a class*
- *Noun Phrases of the Prepositional Phrase share the same head word*
Putting Other Work into Perspective

- **Beacons**: Signal something to the reader [Brooks 83]
  - Not automated

- **Micro Patterns** [Gil and Maman 05]
  - 27 patterns at the level of a class and not methods

- **Extract Topic Words and Phrases** [Maskeri et al 08]

- **Cluster program elements sharing similar words** [Kuhn 07]
  - Not necessarily high level actions

---

- **Summary Comment Generation for Methods** [ASE 10]
  - Identified *individual* statements for a method summary
  - Generated phrases for each statement in isolation
  - This work enables more succinct summaries

Giriprasad Sridhara
Software Maintenance is Difficult

- One of the most expensive phases in a project’s life cycle
  - Consumes 60-90% of the overall resources [Erlikh 2000]

- Why?
  - Program comprehension for maintenance is not easy
    Developers spend more time reading and understanding code than actually writing code [Murphy et al 05]
  - In the industry, typically developers with less experience are assigned maintenance tasks
    - More experienced developers analyze requirements and design a system to meet requirements

- Need to ease program comprehension
Easing Program Comprehension

- Several different techniques.
- Our approach:
  - Reduce the amount of code that a developer has to read or skim
  - Within a method (function)
    - Identify groups of statements that collectively implement a higher level algorithmic action
    - Describe the high level action succinctly using NL phrases
Example of a high level action (in a loop)

```java
for (int x = 0; x < vAttacks.size(); x++) {
    WeaponAttackAction waa = vAttacks.elementAt(x);
    float fDanger = getExpectedDamage(g, waa);
    if (fDanger > fHighest) {
        fHighest = fDanger;
        waaHighest = waa;
    }
}
return waaHighest;
```

Lines 9 to 16 implement a high level action: (Find Maximum)

Target Phrase for Summary: Get weapon attack option object (in vector attacks) with highest expected damage
Sequence as a Single High Level Action

```java
public void reset() {
    removeAll();
    buildGameMenu();
    buildViewMenu();
    buildToolsMenu();
    buildColopediaMenu();
    // --> Debug
    if (FreeCol.isInDebugMode()) {
        add(new DebugMenu(freeColClient));
    }
    update();
}
```
Generating summary comments requires identification of a method’s action and its arguments.

- `os.print(msg);`
  - Action = `get`
  - Theme
  - Secondary Argument

- `string.length();`
  - Action need not be explicit in the method name
  - Theme
  - Secondary Argument

**Developed a Software Word Usage Model (SWUM) to identify linguistic elements in a method**

Background: Variable Lexicalization (ASE 2010)

print(current);

/* print current */

But what is current?

/* print current document */

 UD-Generated

- While reading a method, the context will tell what ‘current’ is, but outside of the body we need to fill in the context.

- Key Insight: Type information typically provides missing context.

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Variable Name</th>
<th>UDGenSubPhrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallFrame</td>
<td>parentFrame</td>
<td>parent call frame</td>
</tr>
</tbody>
</table>

Selectable is an adjective

Frame is not repeated

- Generation:
  - part of speech of words in type and variable names
  - overlap between type & variable name
Background: Phrase Generation Templates (ASE 2010)

- Methods perform actions → Generate verb phrase

**Template for single method call, M(...)**

```plaintext
os.print(msg)  →  UD-GenPhrase  →  /* print message to output stream */
```

Action theme secondary-arguments and get return-type [if M returns a value]
Sequence: Challenges

- To integrate a statement $S_1$ with its successor $S_2$ to build a high level action
  - Determine if $S_1$ is similar to $S_2$.
  - Define notion of Similarity.
Traditional Program Analysis and Text Preprocessing

Method M signature & body

Build Abstract Syntax Tree
Construct Control Flow Graph
Create Def-Use chains

Split Identifiers into Words
[Enslen et al 09]

Expand Abbreviated Words
[Hill et al 08]
public static void main(String[] args) {
    Application app;
    String os = System.getProperty("os.name").toLowerCase();
    if (os.startsWith("mac")) {
        app = new DefaultOSXApplication();
    } else if (os.startsWith("win")) {
        // app = new DefaultMDIApplication();
        app = new DefaultSDIApplication();
    } else {
        app = new DefaultSDIApplication();
    }
    SVGApplicationModel model = new SVGApplicationModel();
    setDifferentAttributes(model);
    app.setModel(model);
    app.launch(args);
}
Clients

- Suggesting more informative method names
  - Example here

- Concern location/Traceability recovery
  - Example here
Clients

- Internal comment generation
  - Example here
- Improving the automatically generated summary (ASE 2010)
  - Example here
- Suggesting more informative method names
  - Example here
- Concern location/Traceability recovery
  - Example here
Abstracting Conditionals: Theme Inference

ELSE IF head word (subject) and predicate of each clause is same THEN

Abstraction = based on which head word predicate object

Lines 6 to 10: “Return movement abbreviation based on movement type”
Transform the clauses (propositions) into corresponding questions using “what” or “which”

Ex: What does os start with?
Use of Relative Clause (Loops)

- To get a more concise phrase
- Relative pronoun: whose or which?
  - Decision criteria
  - Find max
Abstracting Conditionals

Lines 37 to 45: “set property based on which radio button is selected”
Our Other Work in Perspective

- Summary Comment Generation for Java Methods: ASE 2010
  - Focus on identifying *individual* statements important for a summary
  - Generating NL phrases for selected individual statements in isolation

- Parameter comment generation and Integrating with method summaries: ICPC 2011

*Our current work can in fact help in generating more succinct summaries*
Prevalence Results

1.2 million methods from 1000 projects were considered

- **Sequence**
  - Typically not present in very small methods
  - So looked at methods with at least 10 statements
    - 155,289 such methods
  - our prototype detected 17,205 (11%) methods with a sequence high-level action

- **Conditional**
  - 144,562 if-else statements in the methods
  - 58,439 of the had an identified high level action

- **Loop**
  - 82,402 loops iterating over all items in a collection
  - 12,524 of these (15%) were detected as implementing an algorithm pattern

*Our high level action identification heuristics are fairly widely applicable*
Prevalence Results

- **Sequence:**
  - 17, 205/155, 289 methods (11%) methods with a sequence high-level action

- **Conditional**
  - 144,562 if-else statements in the methods
  - 58,439 of the had an identified high level action

- **Loop**
  - 82,402 loops iterating over all items in a collection
  - 12,524 of these (15%) were detected as implementing an algorithm pattern

*Our high level action identification heuristics are fairly widely applicable*
Prevalence Results

- **Sequence**
  - 1.2 million methods from 1000 projects
  - Typically not present in very small methods
  - So looked at methods with at least 10 statements
    - 155,289 such methods
  - Our prototype detected 17,205 (11%) methods with a sequence high-level action

- **Conditional**
  - 144,562 if-else statements in the methods
  - 58,439 of the had an identified high level action

- **Loop**
  - 82,402 loops iterating over all items in a collection
  - 12,524 of these (15%) were detected as implementing an algorithm pattern

*Our high level action identification heuristics are fairly widely applicable*
Phrase Generation Templates (ASE 2010)

- Methods perform actions ➔ Generate verb phrase

**Template for single method call, \( M(\ldots) \)**

```
void print(msg)
```

`/* print message to output stream */`
Our Contribution

Reduce the amount of code that a developer has to read

Develop Heuristics to:

- Automatically detect a group of statements that collectively implement a high level action in a method
- Generate phrases that describes the high-level action as a succinct yet adequate natural language (NL) phrase

Evaluation:

- Human judgements of precision of high level action identification and description
- Prevalence of the identification heuristics across 1000 open source projects
- Potential reduction in reading

Client Tools:

- Illustration of how our technique can improve software maintenance client tools

Outline for the talk
Evaluation Choices

- Utility of high level action identification and description in program comprehension
- High level action identification and abstraction
  - Precision
  - Recall
- Prevalence of the high level actions across many projects
- Utility of high level action identification and description in a client tool
- Reduction in reading detail
Conclusion

❖ First technique to
  • identify code fragments of sequences, conditionals and loops that can be abstracted as a high-level action
  • Synthesize NL description of abstraction

❖ Experienced developers: Good opinion

❖ Good prevalence across 1000 open source projects

❖ Use in automatic comment generation
  • Internal descriptive comments
  • Better Leading Summary Comments

❖ Many other uses
  • Aid Extract Method Refactoring
  • Suggest better method names and local variable names
  • Traceability recovery & Concern location
Our Other Work in Perspective

/* Update linear edge view. If width <= 1, draw line to given graphics2d, else draw polyline to graphics2d */

```java
public void paint(Graphics2D g) {
    update();
    g.setColor(getColor());
    g.setStroke(getStroke());
    setRendering(g);
    int w=getWidth();
    if (w<=1) {
        g.drawLine(start.x,start.y,end.x,end.y);
    } else {
        int dx=w/3+1;
        int dy1=getSourceShift();
        int dy2=getTargetShift();
        int xs[]={ start.x, start.x+dx, end.x-dx, end.x };
        int ys[]={ start.y+dy1, start.y+dy1, end.y+dy2, end.y+dy2 };
        g.drawPolyline(xs,ys,4);
    }
}
```

Our current work can in fact help in generating more succinct summaries
Evaluation Decisions

- Utility of high level action identification and description in program comprehension
  - Studies have already shown that comments (including internal) are useful in comprehension

- High level action identification and abstraction
  - Precision
  - Recall
    - Not easy to measure, prevalence provides an alternative measure

- Prevalence of the high level actions across many projects

- Utility of high level action identification and description in a client tool
  - Future work

- Reduction in reading detail
Summary Comment Generation for Java Methods: ASE 2010

- Focus on identifying *individual* statements important for a summary
- Generating NL phrases for selected individual statements in isolation
- *Our current work can in fact help in generating more succinct summaries*

Comment Studies:

- Many studies have shown the utility of comments (including comments within a method) [Tenny 88, Takang et al 96]
- Programmers tend to comment more in an already well commented system [Marin 05]

Suggesting better method names [Host & Ostvald 09]

- Restricted to action (verb)
- Our work can also help in checking if theme is appropriate
Putting Our Work into Perspective DEL

- **Extract Method Refactoring** [Tsantalis & Chatzigeorgiou 09, Komondoor & Horwitz 03]
  - Our approach would potentially identify candidate fragments not identified through dependencies
  - does not require code transformations to make contiguous
  - can suggest a name for the extracted method

- **Extract & Generate Verb, Noun & Prepositional Phrases** given a method signature [Hill et al 09]

- **Extract Topic Words and Phrases** [Maskeri et al 08]

- **Cluster program elements that share similar words** [Kuhn et al 07]

- **Beacons** [Brooks 83]

- **Micro Patterns** [Gil and Maman 05]

*Orthogonal to current work of identifying high level actions within a method and generating a succinct abstraction of the action*
Our Approach to detecting HL Sequences

Boolean areIntegrable(Statement S1, Statement S2)

VP1 = generateVerbPhraseFor(S1);
VP2 = generateVerbPhraseFor(S2);

if !sameVerb(VP1, VP2)
  return false;

if !sameHeadWord(NP1, NP2) AND !fieldsOfSameClass(NP1, NP2)
  return false;

if present(PP1, PP2) AND !samePreposition(PP1, PP2)
  return false;

return true

S1: contentPane.add(bidPanel);
S2: contentPane.add(endingPanel);

VP1: add bid panel to content pane
VP2: add ending panel to content pane

Same Verb VP1 and Vp2 ? Yes : add

NP1: bid panel
NP2: ending panel
Same Head Word NP1 and Np2?
Yes : panel

PP1: to content pane
PP2: to content pane
Our Approach to detecting HL Sequences

Boolean areIntegrable(Statement S1, Statement S2)

VP1 = generateVerbPhraseFor(S1);
VP2 = generateVerbPhraseFor(S2);

if !sameVerb(VP1, VP2)
    return false;

if !sameHeadWord(NP1, NP2) AND !fieldsOfSameClass(NP1, NP2)
    return false;

if present(PP1, PP2) AND !samePreposition(PP1, PP2)
    return false;

return true

S1: contentPane.add(bidPanel);
S2: contentPane.add(endingPanel);

VP1: add bid panel to content pane
VP2: add ending panel to content pane

subject

VP1: add bid panel VP1: add
VP2: add ending panel VP2: add

Same Verb VP1 and Vp2? Yes : add

NP1: bid panel
NP2: ending panel
Same Head Word NP1 and Np2? Yes : panel

PP1: to content pane
PP2: to content pane
High level actions in loops DEL

```java
for (int x = 0, n = vAttacks.size(); x < n; x++) {
    WeaponAttackAction waa = vAttacks.elementAt(x);
    float fDanger = getExpectedDamage(g, waa);
    if (fDanger > fHighest) {
        fHighest = fDanger;
        waaHighest = waa;
    }
}
```

**Challenges:**

- Developing Identification Templates for common algorithmic steps in a loop
  - Template should be sufficiently general
- Develop heuristics to synthesize a smooth and succinct phrase
Evaluating Automatic Identification and Description of HL Actions DEL

- Effectiveness of identification heuristics
  - Precision
  - Recall
    - Not easy to measure
  - Prevalence
    - Provides an alternative measure to recall

- Usefulness
  - For program understanding
    - Studies have already shown that comments (including internal) are useful in comprehension
  - In a client tool
    - Future work
  - Reduction in reading detail

Giriprasad Sridhara
1.2 million methods from 1000 projects

Typically not present in very small methods

So looked at methods with at least 10 statements

155, 289 such methods

Our prototype detected 17, 205 (11%) methods with a sequence high-level action

High Level Action in Sequence

- Methods without High Level Action in Sequence
- Methods with High Level action
Prevalence Results: Conditionals

- 144,562 if-else statements in the methods
  - 58,439 (40%) of these had an identified high level action

- 17,940 switch statements
  - 4,319 (24%) of these had an identified high level action

![Graph showing the prevalence of if-else, switch, and case statements with high level actions]
Prevalence Results: Loop DEL

- 162,535 loops overall
- 82,402 loops iterating over all items in a collection
- 12,524 of these (15%) were detected as implementing an algorithm pattern

Our high level action identification heuristics are reasonably prevalent
Precision Results: Programmer Opinions on...

- **P1**: The fragment of statements from lines $X$ to $Y$ reflects a high level action that could be expressed as a succinct phrase by a human. i.e.,
  - There are no other statement(s) in the method that you would include in the fragment, AND
  - there are no statement(s) in the fragment that you would exclude from the fragment.

225 Individual Responses (from 75 fragments)
Precision Results: Programmer Opinions on...

- P2: The description represents an abstraction of the block

225 Responses (75 fragments)

- Strongly Agree: 73%
- Agree: 16%
- Neither Agree nor Disagree: 5%
- Disagree: 1%

Good results in identifying and describing high level actions
Can we potentially avoid reading some code?

```java
for (int x = 0, n = vAttacks.size(); x < n; x++) {
    WeaponAttackAction waa = vAttacks.elementAt(x);
    float fDanger = getExpectedDamage(g, waa);
    if (fDanger > fHighest) {
        fHighest = fDanger;
        waaHighest = waa;
    }
}
```

A developer could avoid reading the six statements above and instead read the one phrase below

```java
// Get (Find) weapon attack action object
// (in vectorAttacks) with highest expected damage
for (int x = 0, n = vAttacks.size(); x < n; x++) {
```
Our high level action identification technique leads to a fairly significant reduction in reading detail.
Abstracting Conditionals: Challenges DEL (MERGE WITH SLIDE 16)

- Some challenges similar to finding high level actions in a sequence
- Additionally, in every branch, we can have
  - A return statement
  - Assignment to the same variable
    - The above will not occur in a sequence
- Need to integrate and generate an abstraction for the guarding conditional expressions
Abstracting Conditionals: Return Statement DEL

```java
switch (movementType) {
    case IEntityMovementType.MOVENONE: return "N";
    case IEntityMovementType.MOVEWALK: return "W";
    case IEntityMovementType.MOVERUN: return "R";
}
```

**Not very informative**

**Return String**

**Heuristic:**
IF each return expression is a literal THEN infer a name for the expression based on the action & theme of the enclosing method.

**Theme of the enclosing method:** movement abbreviation

**Return movement abbreviation**

**Strategies for assignment to same variable in each branch**
Describing Conditional Expressions

Heuristic:
IF subject and predicate of each clause is same THEN
Abstraction = based on what subject predicate

Based on what os starts with

Usage of relative pronoun “what”:
Because predicate “starts with” begins with a verb in
the 3rd person singular form

More information in paper
Describing Conditional Expressions

Generate a clause for each conditional expression [ASE 2010]

Heuristic:
IF subject and predicate of each clause is same THEN
Abstraction = based on what subject predicate

Create application, based on what os starts with

Usage of relative pronoun “what”:
Because predicate “starts with” begins with a verb in the 3rd person singular form

More information in paper
Finding High Level Actions in Conditionals

- Integrating statements along the different branches (then and else)
  - Similar heuristics as in finding sequence high level actions
  - Additionally, in every branch, we can have
    - A return statement
    - Assignment to the same variable
    - The above will not occur in a sequence
    - Strategies for the above

- Need to integrate and generate an abstraction for the guarding conditional expressions
Finding High Level Actions in Conditionals: Example

```java
if (os.startsWith("mac")) {
    app = new DefaultOSXApplication();
} else if (os.startsWith("win")) {
    // app = new DefaultMDIApplication();
    app = new DefaultSDIApplication();
} else {
    app = new DefaultSDIApplication();
}
```

Create application, based on what os starts with

Usage of relative pronoun “what”: Because predicate “starts with” begins with a verb in the 3rd person singular form

More information in paper